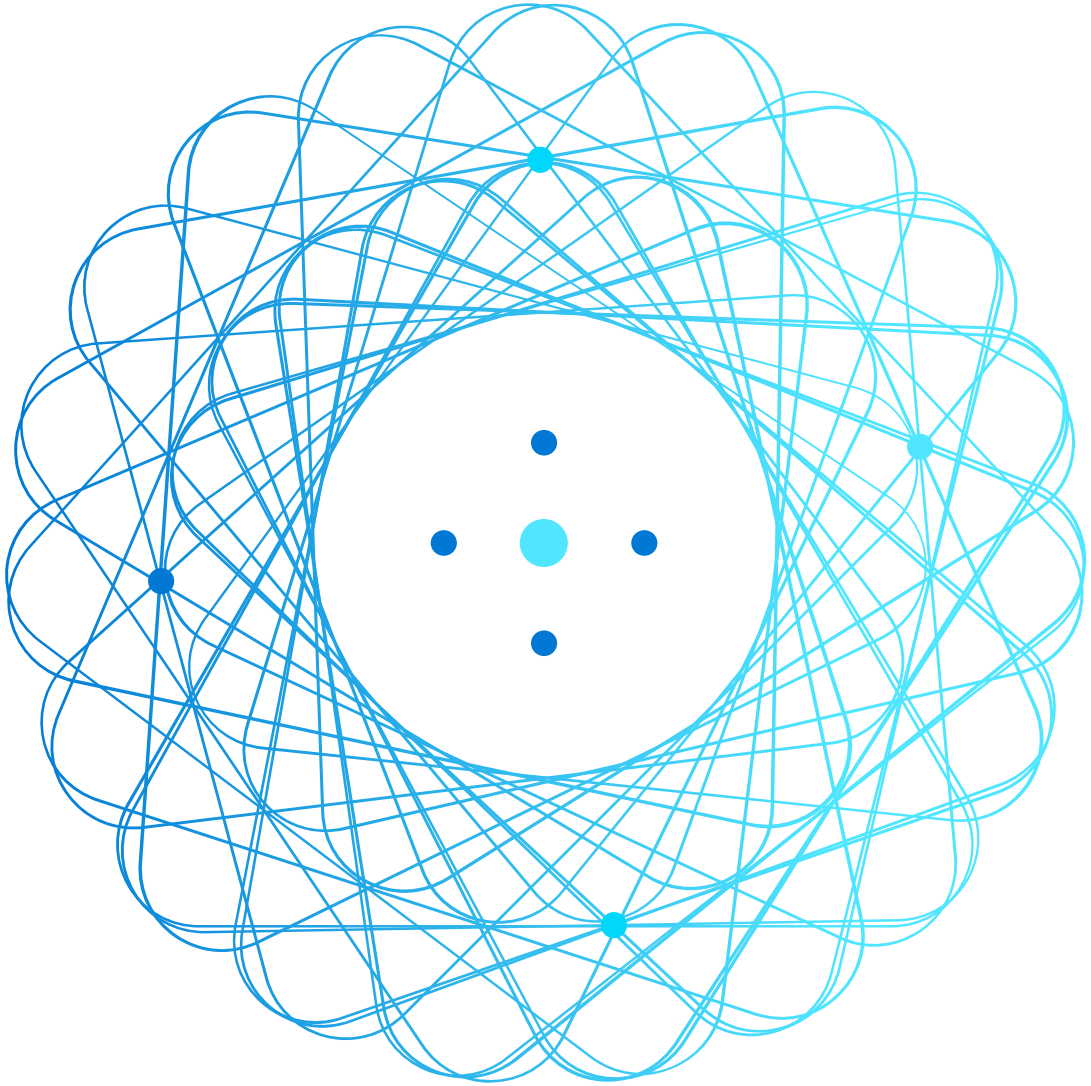
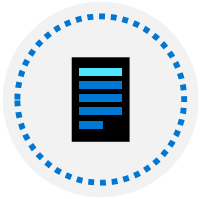
Microsoft Azure

AZ-220T01
Module 07: Azure IoT Edge modules and containers

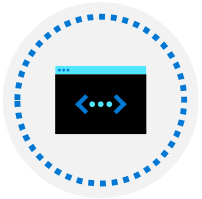


Lesson 1: Learning objectives

Module 7 – Learning objectives

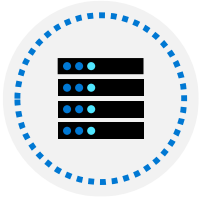


Explain the requirements for building a custom edge module



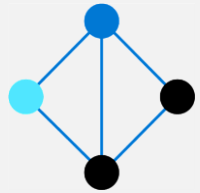
Configure Visual Studio Code for developing containerized modules

Deploy a custom module to an IoT Edge device



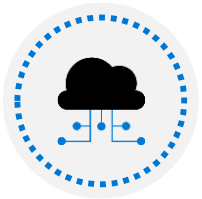
Implement local storage on an IoT Edge device in support of an offline scenario

Lesson 2: Develop custom edge modules

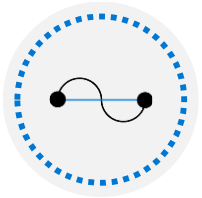


Introduction to IoT Edge module development

IoT Hub primitives

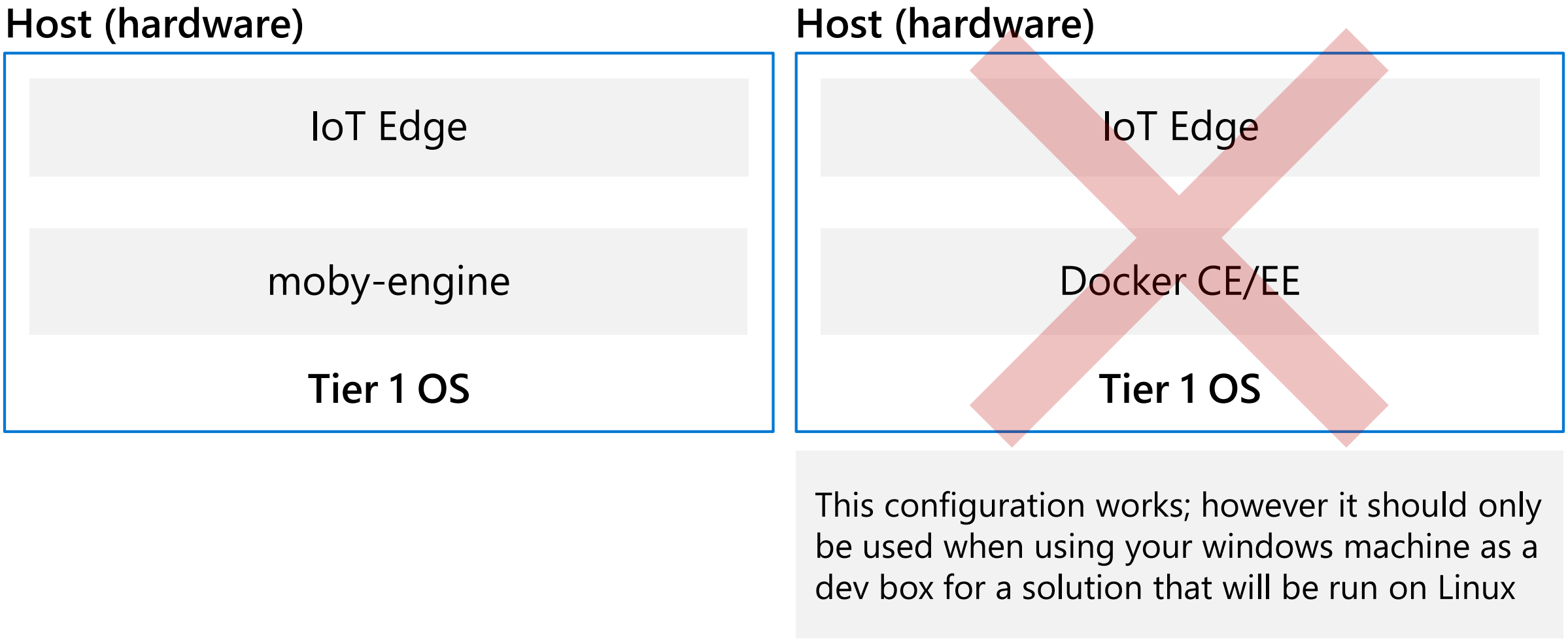


Device-to-cloud messages

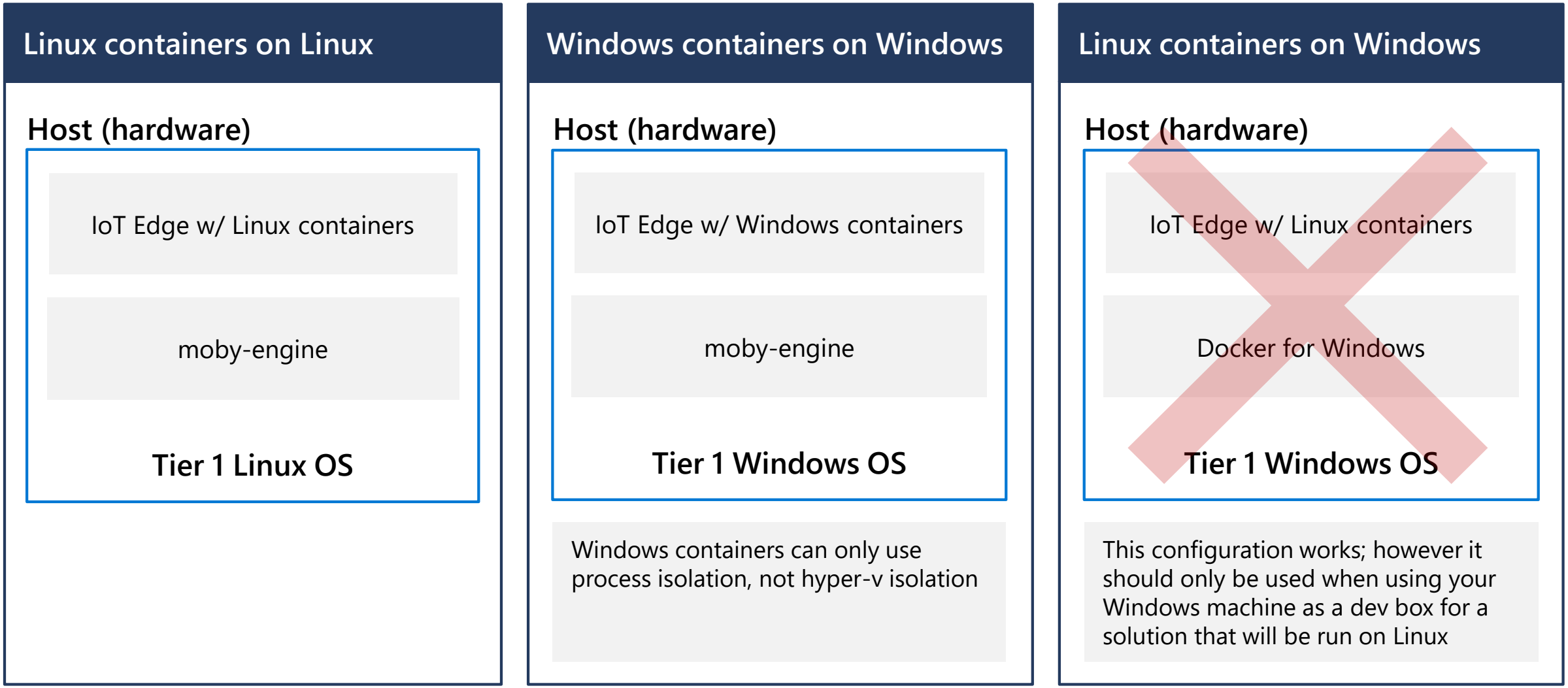


Connecting to IoT Edge hub from a module

Azure IoT Edge supported container engines



Azure IoT Edge supported operating systems



Azure IoT Edge supported virtual machine hosting

Linux containers on Linux, in a VM

Hardware

Host (VM)

IoT Edge w/ Linux containers

moby-engine

Tier 1 Linux OS

Any Linux OS, Windows OS or virtualization platform

Windows containers on Windows, in a VM

Hardware

Host (VM)

IoT Edge w/ Windows containers

moby-engine

Tier 1 Windows OS

Any Linux OS, Windows OS or virtualization platform

Windows containers can only use process isolation, not hyper-v isolation

IoT Edge module development tooling and languages

Linux Targets (AMD64, ARM32)

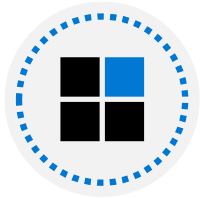
Language	Development tools
C	Visual Studio Code Visual Studio 2017/2019
C#	Visual Studio Code Visual Studio 2017/2019
Java	Visual Studio Code
Node.js	Visual Studio Code
Python	Visual Studio Code

Windows Targets (AMD64)

Language	Development tools
C	Visual Studio Code Visual Studio 2017/2019
C#	Visual Studio Code (*) Visual Studio 2017/2019

(*) no debugging support

Configuring a VS Code development environment



Computer and OS configuration requirements

For Windows modules:

Windows 10 version 1809 (build 17763) or newer

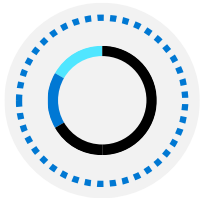
x64 platform

For Linux modules:

Windows 10 (any version), Pro, Enterprise, or Education SKU

x64 platform

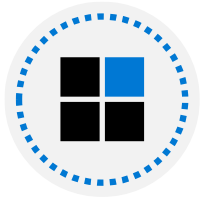
Hyper-V and Container support enabled



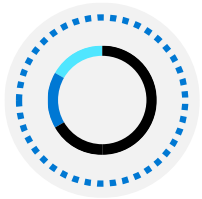
Visual Studio Code configuration requirements

Language-specific tools

Configuring a VS Code development environment



Computer and OS configuration requirements



Visual Studio Code configuration requirements:

Azure IoT Tools

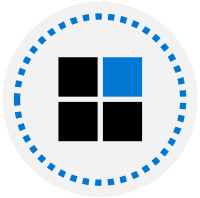
Docker extension

Visual Studio extension(s) specific to the language you're developing in:

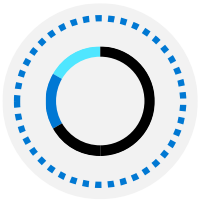
- C#, including Azure Functions: C# extension
- Python: Python extension
- Java: Java Extension Pack for Visual Studio Code
- C: C/C++ extension

Language-specific tools

Configuring a VS Code development environment



Computer and OS configuration requirements



Visual Studio Code configuration requirements

Language-specific tools:

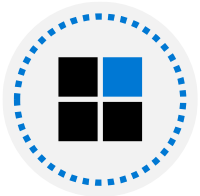
C#, Including Azure Functions: .NET Core 2.1 SDK

Python: Python and Pip for installing Python packages (typically included with your Python installation)

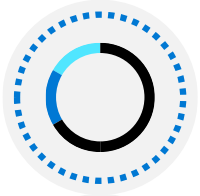
Node.js: Node.js. You'll also want to install Yeoman and the Azure IoT Edge Node.js Module Generator

Java: Java SE Development Kit 10 and Maven. You'll need to set the JAVA_HOME environment variable to point to your JDK installation

Configuring a VS Code development environment

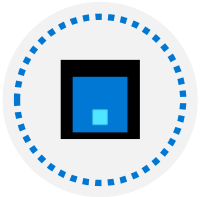


Computer and OS configuration requirements



Visual Studio Code configuration requirements

Language-specific tools



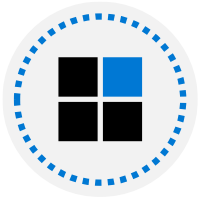
Container tools:

Docker Community Edition on your development machine.

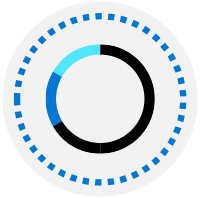
Azure Container Registry or Docker Hub (optional)

Debug tools

Configuring a VS Code development environment

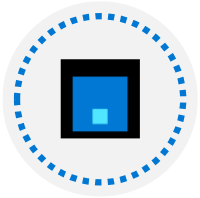


Computer and OS configuration requirements



Visual Studio Code configuration requirements

Language-specific tools

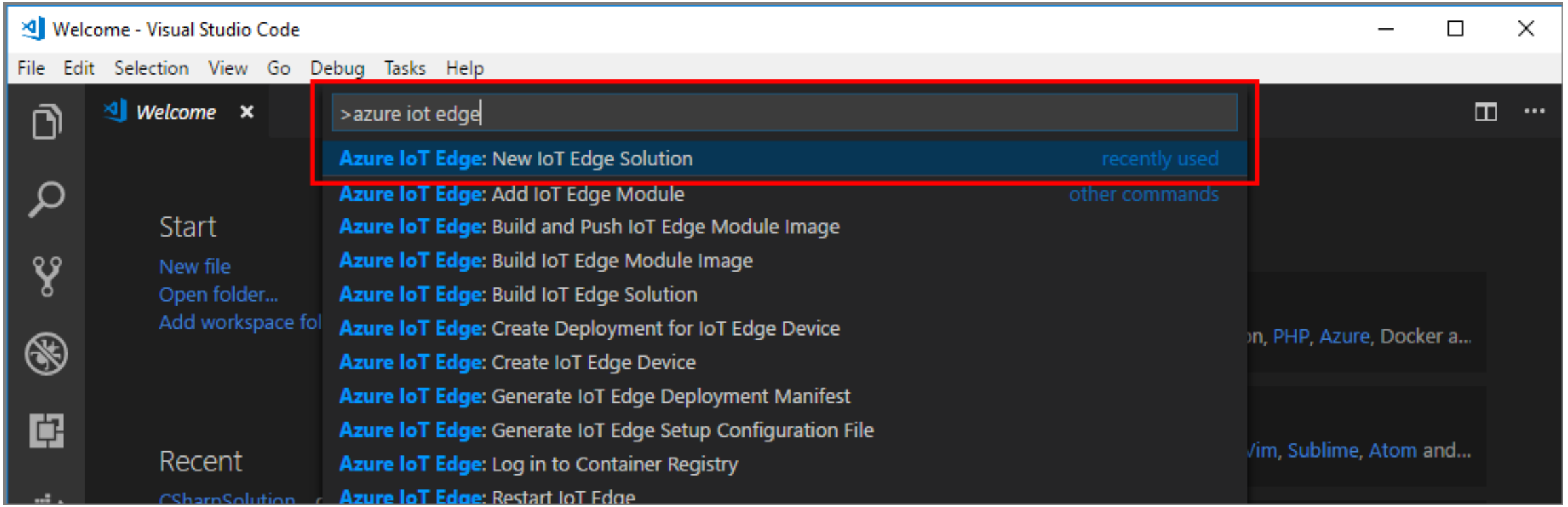


Container tools

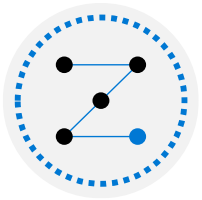
Debug tools:

Except for developing in C, the Python-based IoT EdgeHub Dev Tool should be installed for local debugging and execution

Develop Custom Modules with VS Code



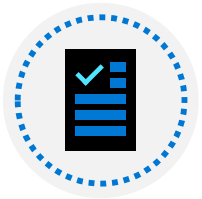
Debugging Modules without the IoT Edge Runtime



Use IoT Edge simulator in place of full infrastructure:

Creates an Edge Hub development (EdgeHubDev) container and a utility container for your module

Does not need an Edge Agent nor a security daemon



Two debug options:

Launch mode – The module is launched from within Visual Studio Code directly; useful for testing a single module

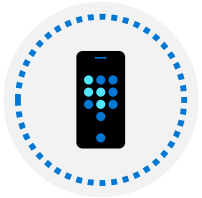
Attach mode – A container and a variation on remote debugging is used; can be used when testing single modules or multiple-module scenarios

Debugging Modules with the IoT Edge Runtime

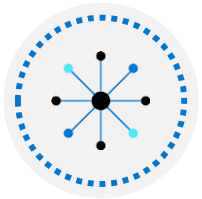
1. Open **deployment.debug.template.json** in Visual Studio Code
2. On the command palette, run **Azure IoT Edge: Build and Push IoT Edge** solution
3. Select your JSON file
4. Locate the appropriate device in your **Azure IoT Hub Devices** list
5. Right-click and select **Create Deployment for Single Device**
6. Select the appropriate debug manifest JSON file
7. Create an appropriate SSH tunnel on your Edge device
8. Connect using remote debugging in Visual Studio Code

Lesson 3: Offline and local storage capabilities

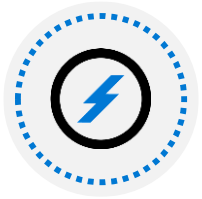
Zero configuration offline capabilities



IoT Edge devices automatically have offline capabilities

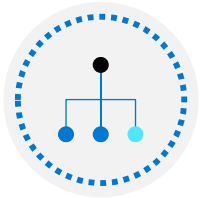


Requires a one-time online connection to an IoT Hub



Allows unlimited duration IoT Edge offline functionality

Extended offline capabilities: What are they?



What Offline Mode handles:

Stores messages for future upstream delivery

Authenticates child devices and modules

Enables cross-device communications that would normally go through IoT Hub

Restrictions and limits:

Requires IoT Edge version 1.0.7 or higher (for full extended offline capabilities)

All regions with IoT Hub *except East US*

Child devices cannot themselves be Edge devices

Offline storage is limited only by message time-to-live in the Edge configuration and storage

Extended offline capabilities: How do you do it?

1. **Configure leaf devices as child devices of the IoT Edge in IoT Hub**

2. **Configure DNS for the container engine on the host**

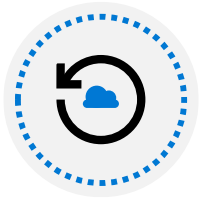
3. **Ensure the IoT Edge Hub can at least once receive its module twin configuration**

4. **Configure optional settings if desired:**

   Time-to-live if the 7200 second (2 hour) default is not appropriate

   Storage on the host instead of inside the Edge Hub container (more on this later in the module)

Intro to Azure Blob Storage on IoT Edge



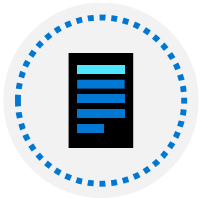
When to use a Blob Storage module:

Data needs to be locally while awaiting post-processing

Limited connectivity to Azure

Reducing blob processing latency through local access

Reducing bandwidth and storage transaction cost by filtering and condensing before sending to Azure



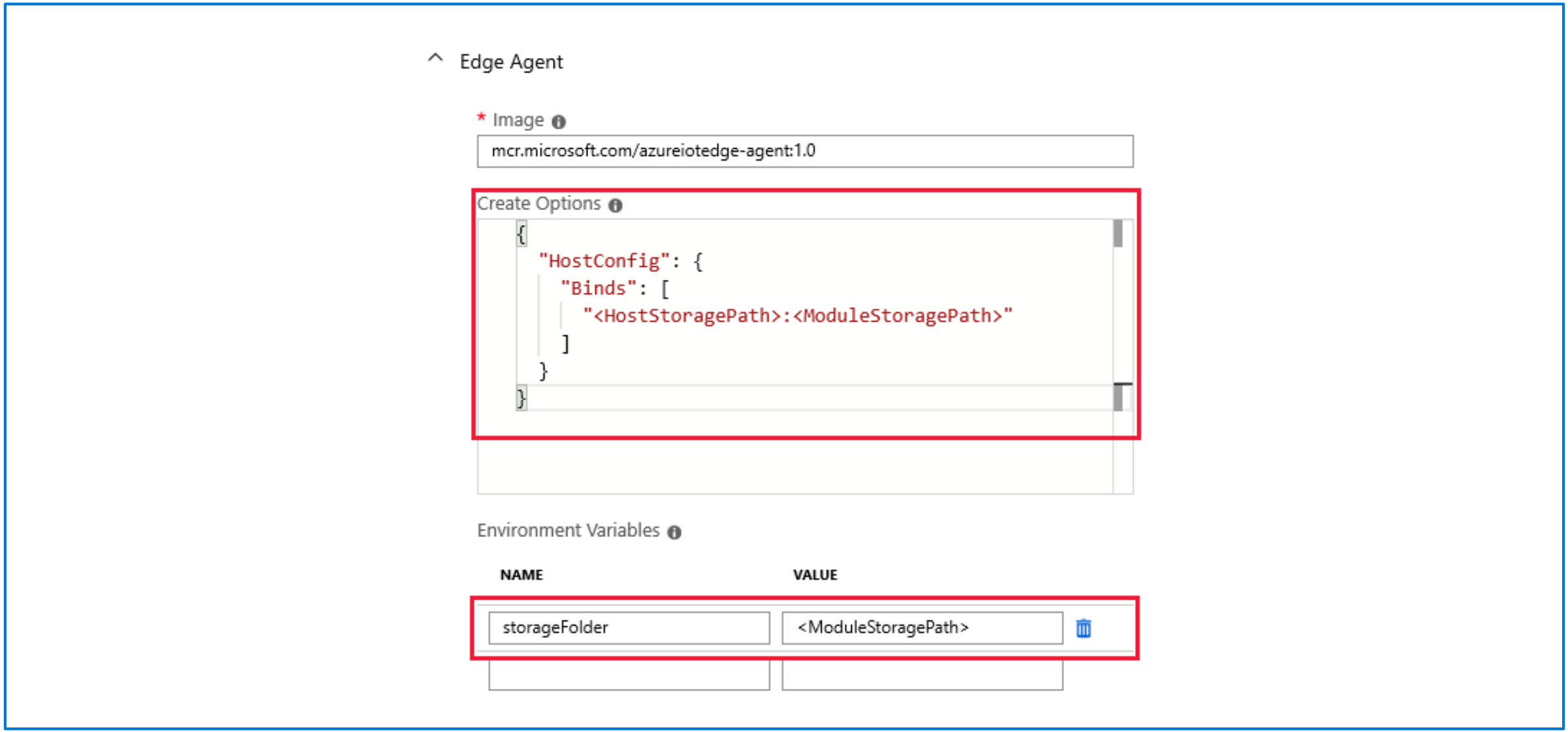
Features of a Blob Storage module:

deviceToCloudUpload – Automatic upload to an Azure storage account

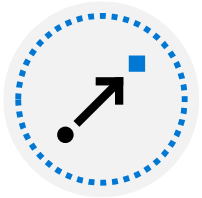
deviceAutoDelete – Clean-up of data from the Edge after a time, including waiting for an upload

Block blob and append Blob Storage (no page blob support)

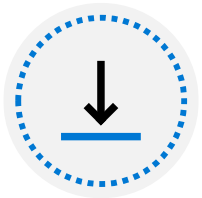
Module access to local storage



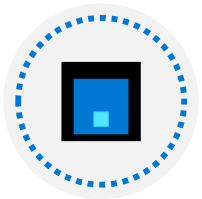
Deploying Azure Blob Storage on IoT Edge



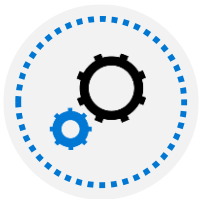
Deploy like any other module



`Image URI: mcr.microsoft.com/azure-blob-storage:latest`



Set **Create Container Options** to be an appropriate JSON document (next slide)



Set **Module Twin Settings** to be an appropriate JSON document (later slide)

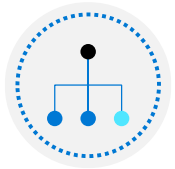
Azure Blob Storage Module create container options

```
{
  "Env":[
    "LOCAL_STORAGE_ACCOUNT_NAME=<your storage account name>",
    "LOCAL_STORAGE_ACCOUNT_KEY=<your storage account key>“
  ],
  "HostConfig":{
    "Binds":[
        "<storage mount>"
    ],
    "PortBindings":{
      "11002/tcp":[{"HostPort":"11002"}]
    }
  }
}
```

Azure Blob Storage Module twin settings

```
{
  "deviceAutoDeleteProperties": {
    "deleteOn": <true, false>,
    "deleteAfterMinutes": <timeToLiveInMinutes>,
    "retainWhileUploading": <true,false>
  },
  "deviceToCloudUploadProperties": {
    "uploadOn": <true, false>,
    "uploadOrder": "<NewestFirst, OldestFirst>",
    "cloudStorageConnectionString": "DefaultEndpointsProtocol=https;AccountName=<your Azure
Storage Account Name>;AccountKey=<your Azure Storage Account Key>; EndpointSuffix=<your end
point suffix>",
    "storageContainersForUpload": {
      "<source container name1>": {
        "target": "<target container name1>“
      }
    },
    "deleteAfterUpload": <true,false>
  }
}
```

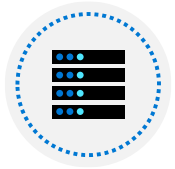
Azure Blob Storage on IoT Edge connectivity



Connecting is very similar to connecting to Azure Cloud

The module configuration includes the appropriate storage account key

This includes being able to use Azure Storage Explorer!



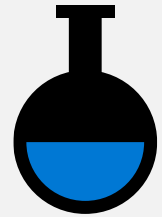
Supported storage operations…

Most of the 2017-04-17 API are supported

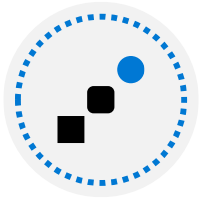
Unsupported operations are generally around ones that don’t make sense in the IoT Edge context, such as setting the blob tier and snapshotting a blob

Event Grid connectivity is available (in Preview)

Lesson 5: Module labs



Module 7 labs

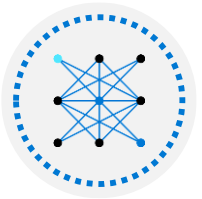


Lab 13: Create and Deploy a Custom Edge Module

You will create the Container Registry

You will create and customize an Edge module

You will deploy modules to Edge device



Lab 14: Implement Restricted Network and Offline Scenarios for IoT Edge

You will setup an IoT Edge Parent device with a Child IoT device

You will configure the IoT Edge device as Gateway and open inbound ports

You will configure the IoT Edge Gateway device Time-to-Live and Message Store

You will connect the Child IoT device to the IoT Edge Gateway and test connectivity and offline support

Lesson 6: Module 7 review questions

Module review: Question 7.1

Which of the following steps is commonly performed when creating a custom IoT Edge module?

Answer A:
Use the Azure portal to create an Azure Container Registry for your Docker container images.

Answer B:
Use the IoT Edge dev tool to configure your production environment.

Answer C:
Use the IoT EdgeHub dev tool to install and configure the IoT Edge runtime.

Module review: Question 7.2

IoT Edge modules share many characteristics with IoT devices.

Which of the following choices accurately describes a characteristic of an IoT Edge module?

Answer A:
It can send device-to-cloud messages.

Answer B:
It can receive cloud-to-device messages.

Answer C:
It can use the file upload feature.

Module review: Question 7.3

A developer for a company has discovered that they are losing data during periods when the local Wi-Fi signal is lost and devices are offline. The developer investigates whether IoT Edge provides a solution for this problem.

Which of the following answer choices accurately describes using Azure Blob Storage for IoT Edge devices?

Answer A:
It uses a blob storage module to provide a block blob storage solution on your IoT Edge device.

Answer B:
It uses the IoT Edge hub module to provide a block blob storage solution on your IoT Edge device.

Answer C:
It uses IoT hub as the blob endpoint for any storage requests that are made by the IoT Edge device.

Module review: Question 7.4

Which of the following choices accurately describes Azure IoT Edge support for extended offline operations?

Answer A:
Full support is provided for all versions of Azure IoT Edge.

Answer B:
After an initial one-time sync with IoT Hub, IoT Edge devices can function indefinitely offline.

Answer C:
The Time-to-Live setting is used to set the time interval that an offline device uses when attempting to reconnect with IoT Hub.